Let's analyze the `mystery` function and determine what it computes. Here is the function for reference:

```
public static int mystery(int a, int b) {
    if (b == 0) return 0;
    if (b % 2 == 0) return mystery(a + a, b / 2);
    return mystery(a + a, b / 2) + a;
}
```

Analyzing the Function

To understand the function, let's go through the steps for `mystery(2, 25)` and `mystery(3, 11)`.

`mystery(2, 25)`

- Initial call: `mystery(2, 25)`
  - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(4, 12) + 2`
    - `mystery(4, 12)`
      - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(8, 6)`
        - `mystery(8, 6)`
          - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(16, 3)` - `mystery(16, 3)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(32, 1) + 16` - `mystery(32, 1)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(64, 0) + 32` - `mystery(64, 0)` returns `0` - Result: `32 + 0 = 32` - Result: `32 + 16 = 48`
          - Result: `48`
      - Result: `48`
  - Result: `48 + 2 = 50`

`mystery(3, 11)`

- Initial call: `mystery(3, 11)`
  - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(6, 5) + 3`
    - `mystery(6, 5)`
      - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(12, 2) + 6`
        - `mystery(12, 2)`
          - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(24, 1)` - `mystery(24, 1)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(48, 0) + 24` - `mystery(48, 0)` returns `0` - Result: `24 + 0 = 24`
          - Result: `24`

        - Result: `24 + 6 = 30`
    - Result: `30 + 3 = 33`

Explanation of the Function

The function `mystery(a, b)` actually performs the multiplication `a * b` using a recursive method similar to the ancient Egyptian multiplication method, also known as peasant multiplication. Here's how it works:

- If `b` is `0`, the result is `0` (base case).
- If `b` is even, it calls itself with `a` doubled and `b` halved.
- If `b` is odd, it adds `a` to the result of the recursive call with `a` doubled and `b` halved.

This is equivalent to decomposing the multiplication into additions.

Modified Function with `+` Replaced by `*`

Let's modify the function:

```
public static int mystery(int a, int b) {
    if (b == 0) return 1;
    if (b % 2 == 0) return mystery(a * a, b / 2);
    return mystery(a * a, b / 2) * a;
}
```

Analyzing the Modified Function

This modified function now performs exponentiation instead of multiplication.

`mystery(2, 25)`

- Initial call: `mystery(2, 25)`
    - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(4, 12) * 2`
        - `mystery(4, 12)`
            - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(16, 6)`
                - `mystery(16, 6)`
                    - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(256, 3)` - `mystery(256, 3)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(65536, 1) * 256` - `mystery(65536, 1)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(4294967296, 0) * 65536` - `mystery(4294967296, 0)` returns `1` - Result: `65536 * 1 = 65536` - Result: `65536 * 256 = 16777216`
                - Result: `16777216`
        - Result: `16777216`

- Result: `16777216 * 2 = 33554432`

`mystery(3, 11)`

- Initial call: `mystery(3, 11)`
  - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(9, 5) * 3`
    - `mystery(9, 5)`
      - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(81, 2) * 9`
        - `mystery(81, 2)`
          - `b` is not `0` and `b % 2` is `0` (even), so it returns `mystery(6561, 1)` - `mystery(6561, 1)` - `b` is not `0` and `b % 2` is `1` (odd), so it returns `mystery(43046721, 0) * 6561` - `mystery(43046721, 0)` returns `1` - Result: `6561 * 1 = 6561`
          - Result: `6561`
      - Result: `6561 * 9 = 59049`
  - Result: `59049 * 3 = 177147`

Summary

- The original function `mystery(a, b)` computes the product `a * b` using a recursive algorithm.
- The modified function with `+` replaced by `*` and `return 0` replaced by `return 1` computes `a` raised to the power of `b` (i.e., `a^b`).